UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Control de actuadores utilizando procesadores embebidos y teléfonos inteligentes

Iván Mangana Fernández
Tutor: Antonio Rovira Moreno
Ponente: Eduardo Boemo Scalvinoni

Julio 2017

Control de actuadores utilizando procesadores embebidos y teléfonos inteligentes

AUTOR: Iván Mangana Fernández
TUTOR: Antonio Rovira Moreno

DSLab
Dpto. Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio 2017

Resumen (castellano)

En este proyecto de fin de grado se va a proponer un sistema de comunicación entre dos terminales. Por un lado tendremos un dispositivo móvil, bajo el sistema operativo Android. Por otro, un sistema embebido que controla una serie de componentes. El sistema embebido estará formado por una placa Arduino con una tarjeta de expansión que le dote de conexión Ethernet. Los elementos a controlar serán: tres leds, un pulsador y un servomotor.

Un punto importante a mencionar es que a lo largo de todo el desarrollo se buscará como objetivo principal que los dos terminales no tengan conectividad directa entre sí. Esto permitirá que el cambio de un terminal por otro no afecte al dispositivo que no ha sido reemplazado. Para ello se hará uso de un servidor intermedio que aloje una base de datos a la que se conectará cada terminal para consultar o modificar el estado de los componentes.

El texto comenzará con un análisis del mercado actual referido a los elementos que se utilizarán y se explicará el motivo de las elecciones tomadas así como de las alternativas que había.

A continuación se expondrán las cuestiones referidas al diseño de la aplicación para posteriormente entrar en el desarrollo, comentando los puntos más técnicos con referencias al código de programación. Tanto las partes de diseño como las de desarrollo pueden estructurarse en tres partes diferenciadas: El servidor, la parte de Android y la parte de Arduino.

Para concluir, se verán y analizarán los resultados finales del sistema y se expondrá el trabajo futuro que puede tener este proyecto. Tanto aplicaciones directas como posibles puntos a mejorar.

Palabras clave (castellano)

Android, Arduino, Servidor Web, Base de datos, Sistema embebido, Control remoto, Aplicación.

Abstract (English)

The aim of this Bachelor Thesis is propose a communication system between two terminals. On the one hand, it will be a mobile phone device that runs under the Android operating system. On the other hand, and embedded system is the responsible for the control of a series of components. The embedded system it will be made up of an Arduino board with an expansion card that provides the Ethernet connection.

An important point to emphasize is that during the entire development the main objective will be that the two terminals do not have a direct connectivity between them. This will allow that a terminal change for another one does not affect the device which has not been replaced. For this, and intermediate server that hosted a database to which each terminal can be connected to consult or modify the situation of the components will be used.

The text will start with a current market analyses related to the elements that will be used. Moreover, the reason for the taken choices and the different options to do it will be explained.

Then, the issues related to the application design will be exposed for subsequently continuous with the development and commenting the most technical points referred to the programming code. Both, the design and the development parts can be structured in three different parts: the server, the Android's part and finally the Arduino's part.

To conclude, the final results of the system will be shown and analyzed and the future work that this project can have will be exposed; direct applications and also, possible points for improvement.

Keywords (English)

Android, Arduino, Web Server, Database, Embedded system, Remote control, Application.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Desde el comienzo de las redes de ordenadores a la red global actual conocida con el nombre de internet han pasado poco más de 50 años. Y en este breve periodo de tiempo los avances al respecto han sido numeroso y su desarrollo exponencial. En 2017, la penetración de internet en la sociedad roza el 50%.

Pero el desarrollo y mejora de internet no permanece estancado. La investigación y mejora por continuar su avance sigue siendo a día de hoy objeto de trabajo por numerosas personas, empresas e instituciones.

Uno de los campos donde más queda por explorar es el de IoT (Internet of things). La idea que yace detrás del IoT se basa en que cualquier objeto cotidiano pueda estar conectado a internet y ser controlado o monitorizado de manera remota.

Según un estudio realizado por la UOC (Universidad Abierta de Cataluña) en los próximos cinco años habrá 50.000 millones de dispositivos conectados a Internet, lo que supone aproximadamente 7 dispositivos por persona de media.

La manera de controlar estos dispositivos ha de ser clara para usuarios de cualquier nivel. Si para cada objeto la manera básica de interactuar con él es distinta, las personas pensarán que esta conectividad tiene más contras que pros. Una de las soluciones más extendidas en la actualidad es el de emplear un dispositivo móvil o Smartphone ya que es un elemento con gran presencia en la sociedad y no supone un gasto extra de cara al usuario; pues (en general) quien quiera usar objetos bajo el concepto de IoT ya poseerá uno. Otras alternativas pueden ser el uso de wearables como smartwatchs o pulseras inteligentes.

El proyecto forma parte de la línea de investigación presente en el DSLab orientada al control remoto de distintos sistemas mediante el uso de dispositivos móviles. Pretende sentar unas bases en lo que a comunicación entre dos terminales se refiere y ser un punto de partida para el diseño de futuras aplicaciones.

1.2 Objetivos

El objetivo principal de este TFG será el de realizar un diseño y una implementación de un sistema que permita controlar los actuadores de un sistema embebido a través de un teléfono inteligente o Smartphone.

La idea principal que se ha tenido en cuenta a la hora de realizar el diseño es que las dos partes que intervienen en el proceso (el Smartphone y el sistema embebido) sean totalmente independientes entre sí. Esto quiere decir que si uno de los dos terminales es reemplazado por otro no se necesario modificar nada del dispositivo que no ha sido reemplazado. Se empleará un servidor remoto que actuará como conexión entre ambos y gestionará los datos y la información que se quiere compartir.
Para ello diferenciaremos la estructura del sistema en tres partes:

- Smartphone con sistema operativo Android y conexión a internet.
- Servidor que soporte el lenguaje PHP y la gestión de bases de datos.
- Sistema embebido con conexión a internet

Para probar que el sistema de comunicación funciona se realizará un pequeño montaje hardware formado por tres leds, un pulsador y un servomotor.

Desde el terminal móvil se podrán controlar el estado de los leds (encenderlos o apagarlos) y la posición de giro del servomotor. Desde Arduino se controlará el estado del pulsador, pudiendo este estar activo o inactivo. A parte de controlar el estado de algunos elementos, la aplicación móvil también mostrará al usuario de forma gráfica el estado actual de todos los elementos.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

1. **Introducción.** Donde se explicarán los motivos y los objetivos de este proyecto.
2. **Estado del arte.** Se analizarán varias alternativas a la hora de realizar el proyecto. También se verán ejemplos de proyectos que siguen los objetivos de éste.
3. **Diseño.** Se explicarán las decisiones tomadas en cuanto al diseño de la aplicación se refiere.
4. **Desarrollo.** Se enunciarán las cuestiones referentes al desarrollo de cada parte del proyecto. Se verán las soluciones a los problemas y se analizará el código realizado.
5. **Integración, pruebas y resultados.** Se verán los resultados obtenidos, testeando la funcionalidad del proyecto.
6. **Conclusiones y trabajo futuro.** En este capítulo se razonarán cuáles han sido los resultados obtenidos y se profundizará en como continuar en un futuro el proyecto.

2 Estado del arte

A lo largo de este apartado veremos los distintos elementos que podemos usar para realizar el proyecto y argumentaremos el motivo a la hora de decantarnos por uno u otro. Posteriormente especificaremos algunos de los proyectos llevados a cabo con las premisas y logros que queremos alcanzar.

2.1 Terminal Móvil (Smartphone)

En los últimos años, el mercado de los Smartphone en lo que a sistema operativo se refiere ha ido evolucionando hasta derivar en dos grandes competidores que se han adueñado del mercado. Android e iOS. Los sistemas operativos propiedad de Google y de Apple respectivamente ocupan más del 95% de cuota de mercado.

De entre los dos destaca Android con un 66,71%. Además, este dato se incrementa en España donde Android sobrepasa el 80% de cuota de mercado

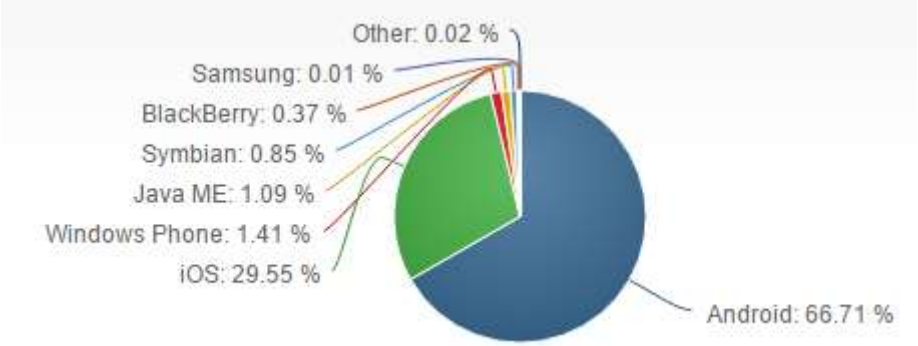


Figura 2-1: Sistemas operativos móviles a nivel mundial.

Todos estos datos, sacados de la web de estadísticas NetMarketShare y con fecha de 2017, hacen decantarse en este proyecto por el desarrollo bajo el sistema operativo Android. No obstante, sería interesante contemplar el desarrollo de la aplicación bajo el sistema operativo de iOS.

2.2 Servidor

Como se ha comentado en los objetivos y se desarrollará en profundidad más adelante, la comunicación entre los dos terminales se realizará mediante un servidor intermedio. Dicho servidor debe permitir el alojamiento de bases de datos mediante las cuales los dos terminales intercambiaran información.

Dada la gran alternativa actual en cuanto a servidores remotos se refiere se ha optado por utilizar uno en vez de montar un servidor privado propio para el proyecto.

Dentro de las distintas compañías que ofrecen este tipo de servicios se ha tenido en cuenta que cumplan los siguientes requisitos:

- Soportar el lenguaje PHP sobre el que se lanzaran las peticiones desde la aplicación Android.
- Incluir bases de datos, a ser posible gestionado bajo MYSQL.
- Permitir conexión mediante FTP para subir los archivos necesarios.

Los tres principales servicios que se han encontrado y que además no tienen ningún coste son los siguientes:

Figura 2-2: Logo de 000webhost

000webhost

Incluye dos bases de datos MYSQL, dos dominios distintos, 1000MB de espacio en disco y un ancho de banda mensual de 10 GB de ancho de banda mensual. Aunque estos dos últimos puntos no son relevantes.

Figura 2-3: Logo de Freehostia

Freehostia

Incluye una base de datos MYSQL, cinco dominios distintos, 250MB de espacio en disco y un ancho de banda mensual de 6 GB.

Figura 2-4: Logo de Hostinger

Hostinger

Incluye dos bases de datos MYSQL, un dominio, 2000MB de espacio en disco y un ancho de banda mensual de 100 GB.
Es la opción por la que nos hemos decantado a la hora de hacer el proyecto. El factor más determinante a la hora de su elección ha sido el cómodo panel de control, con acceso a todas las herramientas de manera clara e intuitiva.

2.3 Sistema embebido

Se conoce por sistema embebido o empotrado como aquel sistema electrónico diseñado para realizar tareas concretas y dedicadas. Una de sus características más importantes es que suele ser fundamental que los sistemas den respuesta en tiempo real, al contrario que un sistema de escritorio general (PC). Además, sus recursos suelen ser más limitados.

Dentro de los sistemas embebidos se ha tenido en cuenta tres modelos con los que se puede realizar el proyecto.

2.3.1 Raspberry

Raspberry es un computador de tamaño reducido desarrollado en Reino Unido. Pese a ser un producto registrado, es de software libre, permitiendo el poder instalar un sistema operativo bajo el que funcione. Su coste es muy bajo y presenta características idóneas para el proyecto como su fácil conexión a la red y bajo consumo, pero otros puntos negativos han llevado a declinar esta opción. El principal es que no soporta directamente entradas analógicas.

Figura 2-5: Placa Raspberry

2.3.2 Nanode

Se trata de una placa de circuito impreso que tiene un microcontrolador y distintos puertos de entrada/salida. Tiene un diseño orientado a la red y es de código libre. Sus puntos fuertes radican en su coste reducido y su orientación al mundo de IoT.

Figura 2-6: Placa Nanode

2.3.3 Arduino

Al igual que Nanode, Arduino está formado por una placa con un microcontrolador. Posee distintos puertos de entrada/ salida y permite la conexión de shields o placas de expansión. De hecho, puede considerarse como un antecesor de la placa Nanode. Tiene un software de código abierto.

Figura 2-7: Placa Arduino UNO

Esta ha sido la placa seleccionada para realizar el proyecto. Se ha impuesto a la placa Nanode por el hecho de que tiene una mayor cuota de mercado, haciendo que el número de librerías desarrolladas con las que poder trabajar y el respaldo de la comunidad sea mucho mayor.

Dentro de las placas de Arduino existen varios modelos que se diferencian en algunos puntos concretos como puede ser el número de pines disponibles, el microcontrolador integrado o la frecuencia de trabajo.

Para el proyecto se va a realizar la conexión de arduino con la red de internet mediante Ethernet. Para ello hay dos alternativas, Arduino Ethernet que incluye en una sola placa los elementos que permiten la conexión Ethernet junto a los componentes de arduino o la placa de expansión Ethernet Shield 2. Esta últma alternativa será la llevada a cabo en el proyecto.

La placa Ethernet Shield 2 se conecta a una placa de Arduino compatible (en este caso Arduino UNO) dotando de la funcionalidad de Ethernet a ésta.

Figura 2-8: Placa Ethernet Shield 2

2.4 Arduino y Android: Proyectos

Como se acaba de ver, los dos terminales entre los que se producirá la comunicación en este proyecto estarán gobernados por Arduino y por Android. A continuación, vamos a ver algunos proyectos que combinan estos dos sistemas.

2.4.1 Control de persianas

El usuario Johanmoberg presentó en la página Instructables un proyecto para controlar las persianas de su casa a través de un terminal con Android. El sistema está formado por un dispositivo impreso en 3D con un motor que es controlado mediante Arduino. La conexión entre Android y Arduino es llevada a cabo mediante Bluethooth Además cuenta con un temporizador que permite prefijar cuando se quieren subir o bajar las persianas. Su proyecto ha sido visualizado por más de 18.000 personas.

Figura 2-9: Prototipo del proyecto de Johanmoberg

2.4.2 Motorización de fluidos

Smart Gallow es el nombre de este proyecto presentado en abril de 2016. Una placa de Arduino MKR1000 controla un sensor no invasivo que permite parametrizar distintos aspectos de un fluido. Inicialmente ideado para instalarse en garrafas de agua, almacena datos como el nivel de agua en un servidor Azure de Microsoft que pueden ser observados en tiempo real a través de un Smartphone o de una página web.

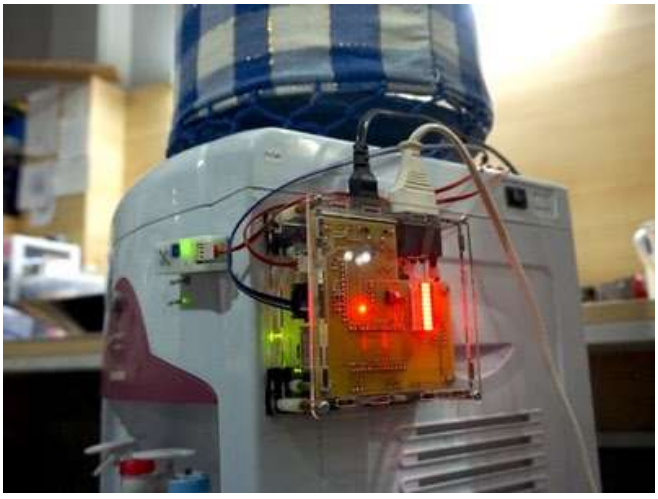


Figura 2-10: Proyecto Smart Gallow

2.4.3 ExControl

ExControl es una empresa con sedes en Madrid y Cáceres dedicada a la evaluación, desarrollo e instalación de soluciones domóticas. Emplean Hardware como Arduino en sus soluciones y adaptan los proyectos a las exigencias del cliente. A parte del sistema domótico en sí, también elaboran las aplicaciones para Smartphone o Tablet con las que se puede controlar todo el sistema.

Entre los campos de domótica que trabajan está a iluminación controlada, el manejo de toldos y persianas, control de accesos en puertas, video vigilancia o sistemas de riego.

Figura 2-11: Logo de ExControl

2.4.4 Videovigilancia

Existen numerosos proyectos de este ámbito. Uno de ellos es el propuesto en la página de Makeuseof. Una placa Arduino controla un servomotor que permite la rotación de una cámara de vigilancia. Los archivos de video son guardados en una Raspberry pi, pero el proyecto podría adaptarse para guardarlos en un servidor dedicado y acceder a ellos a través de nuestro Smartphone.

Figura 2-12: Prototipo proyecto videovigilancia Makeuseof

3 Diseño

En esta subsección se verán las decisiones de diseño tomadas a lo largo del proyecto y los aspectos más relevantes en cuanto a la funcionalidad de los sistemas sin entrar a profundizar en detalles técnicos.

3.1 Servidor

El servidor en el que se almacena la base de datos y se alojan los scripts de PHP que son necesarios para la comunicación tiene la IP 31.170.167.188 y su dirección web asociada es *http://actuadorandroid.hol.es*. A continuación se detallan los aspectos más destacados del servidor.

3.1.1 Base de datos: MySQL

MySQL es el sistema empleado para gestionar la base de datos. La manera más cómoda de acceder a la base de datos es a través de phpMyAdmin. Ingresando a ella a través del panel de control de Hostinger e introduciendo los datos de acceso.

Figura 3-1: Formulario de acceso a la base de datos

Una vez dentro podremos añadir tablas, modificarlas, actualizar datos… todo de manera intuitiva. No obstante, también dispone de una terminal en la que escribir directamente los comandos SQL necesarios.

A continuación, se muestra la tabla empleada en el funcionamiento de este proyecto, con el sistema de doble detección detallado anteriormente.

	Identificador	LedR	LedV	LedA	Servo	Pulsador
Editar Copiar Borrar	Android	1	0	1	90	1
Editar Copiar Borrar	Arduino	1	0	1	90	1

Tabla 3-1: Tabla de la base de datos para el proyecto

Cada una de las dos filas de la tabla corresponde a uno de los terminales, identificados mediante el parámetro de la primera columna “Identificador”. Las tres siguientes columnas pertenecen a los tres leds del circuito. La penúltima es la que recoge el valor que se le quiere dar a la posición del servomotor. Por ultimo esta la columna del pulsador.

3.1.2 Acceso FTP

Para acceder al servidor existen varios métodos como SSH o FTP. En este caso, la conexión s hará por este segundo método, FTP. Principalmente por su sencillez y comodidad.

Como cliente FTP hemos usado el programa Filezilla y la configuración necesaria conectarse al servidor son los siguientes:

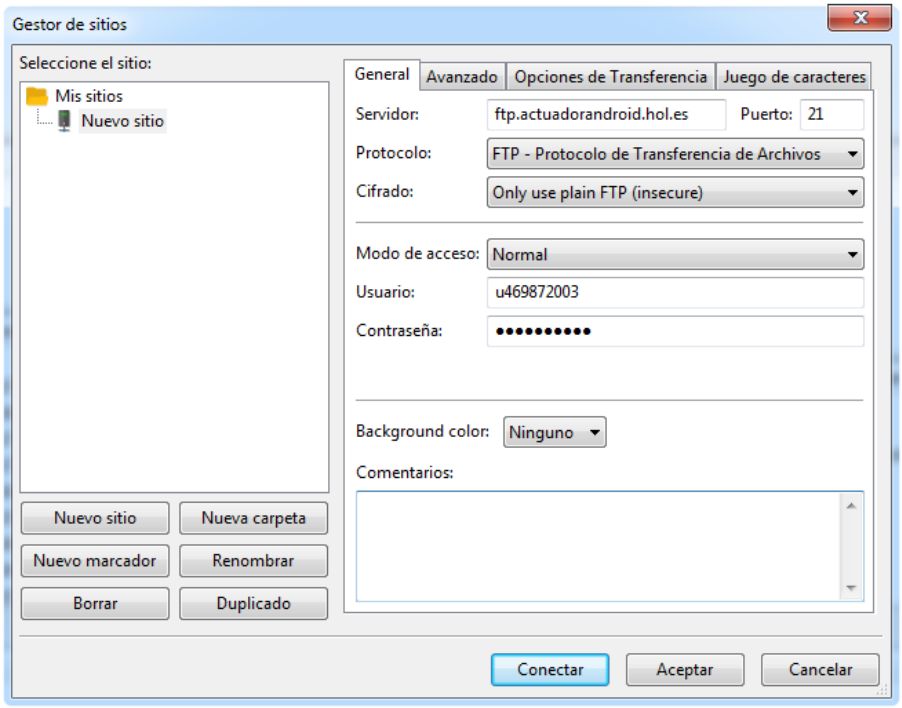


Figura 3-2: Datos conexión FTP

- **Servidor:** Está formada por la dirección web precedida por el comando ftp. También sería válido emplear la dirección IP del servidor.
- **Puerto:** Los puertos típicos en una conexión FTP son el 20 y el 21. Hostinger nos obliga a usar el puerto 21 para el acceso.
- **Protocolo:** Esta SFTP y FTP. Emplearemos el segundo.
- **Cifrado:** Pese a que es el menos fiable en aspectos de seguridad, es el requerido por el servidor para asegurar que las transferencias siempre sean correctas.
- **Modo de acceso:** Normal o por defecto serán los empleados.
- **Usuario y contraseña:** El usuario es asignado por el propio servidor, la contraseña puede ser elegida por el usuario.

3.1.3 Sistema de doble tabla

Uno de los puntos que se quieren garantizar en la realización del proyecto es la fiabilidad de los datos mostrados en la aplicación. Si se quiere hacer un control a distancia de la placa de Arduino es fundamental que los datos mostrados al usuario a través de la pantalla del terminal móvil sean exactos y precisos.

Con el siguiente ejemplo se podrá ver más claro:

> La aplicación permite manejar varios leds de Arduino de forma remota y además muestra por pantalla el estado (encendido o apagado) de cada led. Una vez que se da a la aplicación la orden de cambiar un estado de un led, esta enviará la orden a la base de datos para que Arduino pueda ejecutarla y a continuación quedará en espera de recibir la confirmación de que la acción se ha realizado, y poder entonces a proceder a actualizar la pantalla de la aplicación mostrando el nuevo estado de los leds.

Como se ha explicado en el ejemplo, el procedimiento utilizado es el de esperar a que Arduino active el led y lo notifique para informar al usuario de que el proceso ha sido realizado correctamente.

Con el fin de que este sistema funciones se ha utilizado un método de duplicación de tablas.

Figura 3-3: Esquema de comunicación Terminal-Base de datos

Cada elemento a controlar tendrá dos registros en la base de datos, una instancia correspondiente a Android y otra a Arduino. Si desde un terminal se modifica el estado de un componente, el propio terminal no asumirá que la modificación ha sido completada hasta que el otro terminal lo notifique en la base de datos.

Por tanto, un terminal podrá escribir los datos del estado de un componente en la fila de la tabla correspondiente a su instancia. Por el contrario, para leer el estado en el que dicho componente se encuentra tendrá que hacerlo en la fila correspondiente al otro terminal.

3.2 Arduino

A continuación, se revisan los aspectos de diseño relacionados con Arduino. Tanto de la placa como de los componentes añadidos que forman el sistema. No se tendrán en cuenta detalles técnicos ni referencias a código. Para ver estos aspectos habrá que ver el punto 4 de esta memoria.

3.2.1 La placa

Como se vio anteriormente, se ha usado como sistema embebido la placa Arduino UNO complementada con la tarjeta de expansión Ethernet Shield 2.

La placa de expansión Ethernet se conecta sobre el Arduino UNO. Tanto los pines analógicos como los digitales de Arduino UNO salen a través de la tarjeta de Ethernet por tanto no ha sido necesario realizar ningún cambio en las tarjetas ya que la compatibilidad entre ellas es total.

Los cables que necesita Arduino son dos. El de Ethernet, conectado directamente al router del que dependerá la conexión a internet y el de alimentación, que se realizó a través de un USB tipo B y que se conectara a un ordenador. Mediante este cable se realiza tanto la programación como la alimentación de la placa.

También sería posible alimentar Arduino mediante una pila de 9V con el adaptador que podemos ver a continuación.

Figura 3-4: Adaptador alimentación Arduino

Existen dos tipos de tarjetas de expansión Ethernet. Una que incorpora la alimentación vía Ethernet y otra que no lo hace. La utilizada es del segundo tipo ya que su precio es menor y no es necesario utilizar esta característica porque podemos alimentar la placa por el cable de programación.

3.2.2 Componentes

Los elementos que forman parte de nuestro sistema son los siguientes:

- **Tres leds.** Uno rojo, uno verde y uno azul.
- **Un pulsador.** Se considerará activo cuando permanezca pulsado.
- **Un servomotor.** Se controlará su posición.

Salvo el servomotor, el resto de componentes se colocarán sobre una PCB. Esto se ha realizado así por su sencillez y facilidad de montaje.

El esquema completo es el siguiente. Pasaremos posteriormente a analizar cada parte.

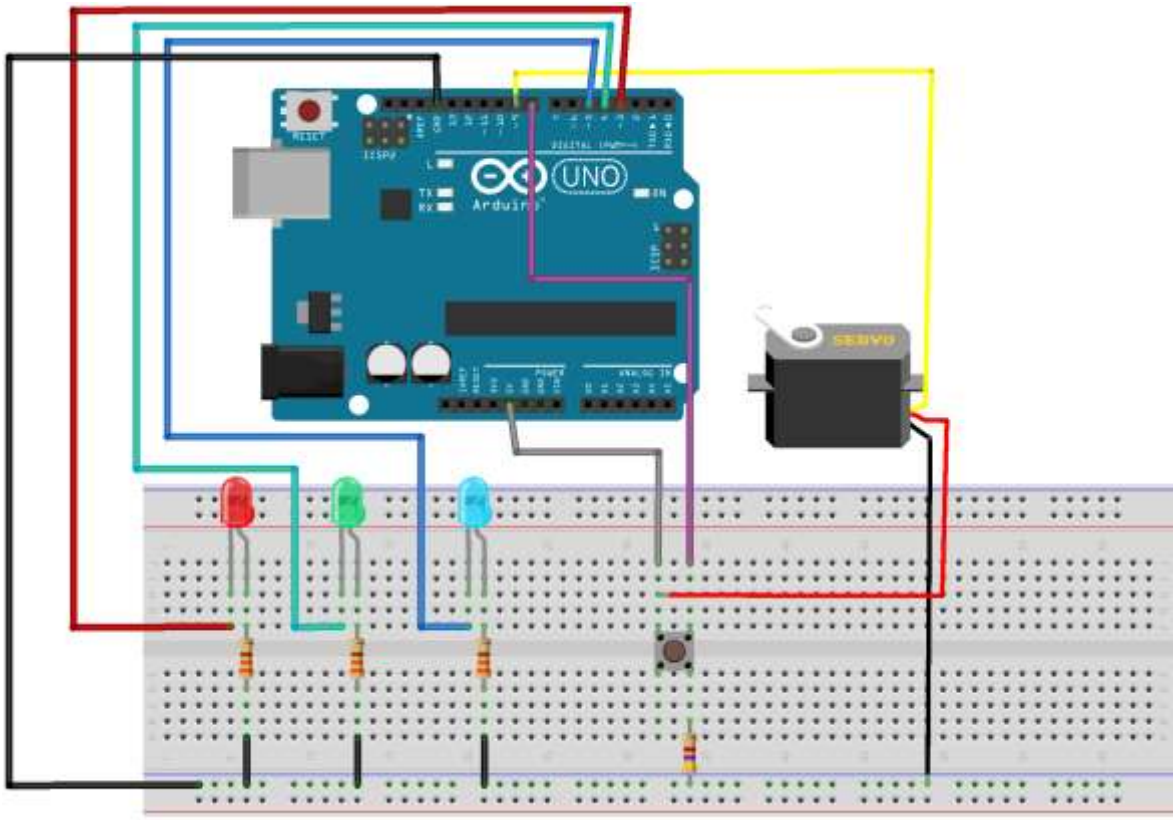


Figura 3-5: Esquemático del montaje Arduino

Leds

Cada uno de los tres leds va conectado a una salida digital de Arduino. Llevan una resistencia en serie de 330 Ω. Gracias a esta resistencia se limita la corriente que pasa por el led evitando que pueda llegar a quemarse. La resistencia conecta directamente con GND.

Pulsador

El pulsador es alimentado mediante el pin de 5v de Arduino. El otro extremo se ha conectado al pin 9 para poder ser leído. El terminal conectado al pin 9 tiene una resistencia de Pull-down de 10KΩ para no cortocircuitar la entrada y la salida cuando el interruptor se cierre.

Servomotor

El servomotor tiene 3 pines. Uno se conecta a la alimentación de 5v, otro a GND y el último a un pin PWM por donde recibirá las instrucciones vía Arduino. La posición en la que se puede colocar el servomotor variará entre 10° y 99°.

Este es el aspecto final del montaje. Salvo el servomotor, todos los elementos han sido conectados sobre una PCB.

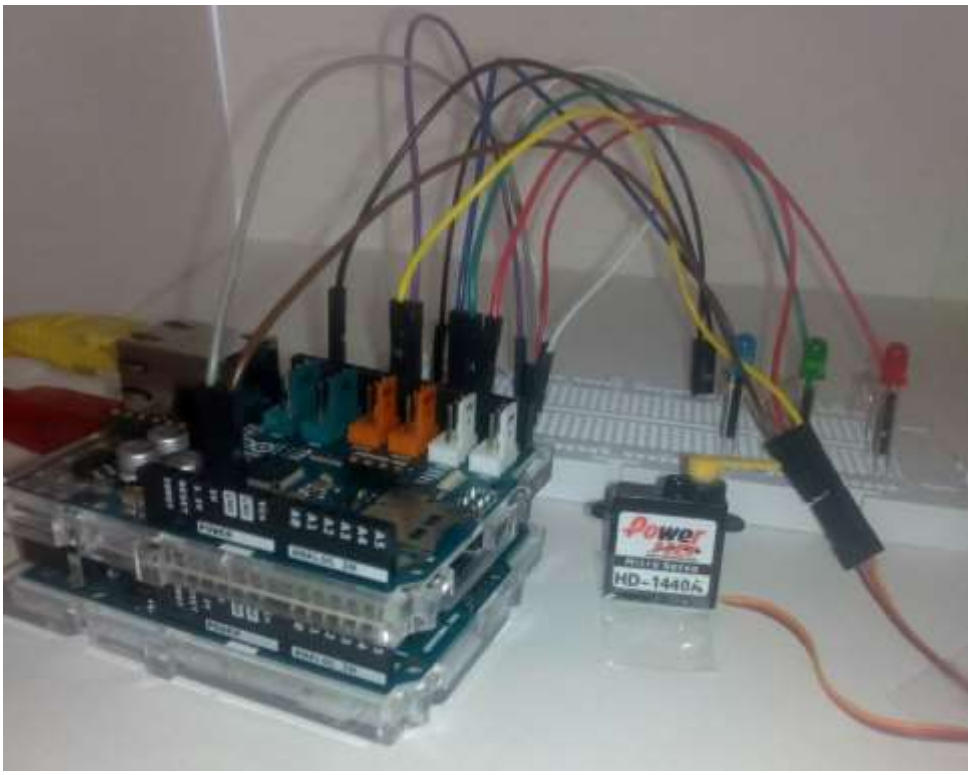


Figura 3-6: Montaje Arduino y PCB

3.3 Android

La aplicación creada para el sistema operativo Android lleva el nombre de Androidino. A continuación, se detallarán los principios de diseño tomados y el aspecto funcional de la misma. Para ver en detalle el funcionamiento interno de la aplicación será necesario acudir al apartado 4 (Desarrollo).

Figura 3-7: Logo de la aplicación Androidino

La principal premisa a la hora de desarrollar la App ha sido que sea lo más simple e intuitiva posible. El aspecto visual no ha sido un punto prioritario. Simplemente se ha buscado que fuese clara para el usuario.

La aplicación solo tiene una pantalla y no es necesario realizar ninguna configuración previa. Esto es así porque el objetivo final no es lanzar una App al mercado sino desarrollar un sistema de comunicación entre dos terminales que de paso a una futura expansión del proyecto agregando distintas funcionalidades o haciéndolo compatible con otro tipo de sistemas (se detallará en la sección 6, Trabajo futuro).

La aplicación está en idioma español, aunque no es necesario saber la lengua para poder utilizarla. El único requisito para su funcionamiento es el de la conexión a internet. Esta conexión puede ser llevada a cabo de cualquier forma que sea permitida por el dispositivo; WiFi, 3G, 4G, HSPA…

La herramienta bajo la que se ha programado ha sido Android Studio, lanzada por Google en 2014, es el IDE oficial para el desarrollo de aplicaciones Android. El programa es gratuito y multiplataforma.

3.3.1 Versión Android

La primera decisión que se ha tenido que tomar antes de comenzar con la programación de la App ha sido la de establecer que versiones de Android puede soportar. Dependiendo de la versión la manera en la de realizar algunas implementaciones cambiará, como es el caso de la conexión a internet que veremos más adelante.

Google ofrece una tabla de gran ayuda a la hora de tomar la decisión. En ella se muestra el porcentaje de sistema operativo que tienen los dispositivos activos en su tienda.
A junio de 2017 estos son los valores:

Versión	Nombre	API	Distribución	Distribución acumulada
2.3	Gingerbread	10	0.8%	100%
4.0	Ice Cream Sandwich	15	0.8%	99.2%
4.1	Jelly Bean	16	3.1%	98.4%
4.2	Jelly Bean	17	4.4%	95.3%
4.3	Jelly Bean	18	1.3%	90.9%
4.4	KitKat	19	18.1%	89.6%
5.0	Lollipop	21	8.2%	71.5%
5.1	Lollipop	22	22.6%	63.6%
6.0	Marshmallow	23	31.2%	40.7%
7.0	Nougat	24	8.9%	9.5%
7.1	Nougat	25	0.6%	0.6%

Tabla 3-2: Distribución versiones de Android

Viendo los valores que nos muestran se ha tomado la decisión de realizar el diseño compatible con la API 15 (y posteriores). La aplicación será compatible con el 99.2% de los móviles activos bajo un sistema operativo de Android.

3.3.2 Pantalla principal

A continuación, se verá la interfaz de la aplicación. Se detallarán sus partes y componentes así como todo su funcionamiento.

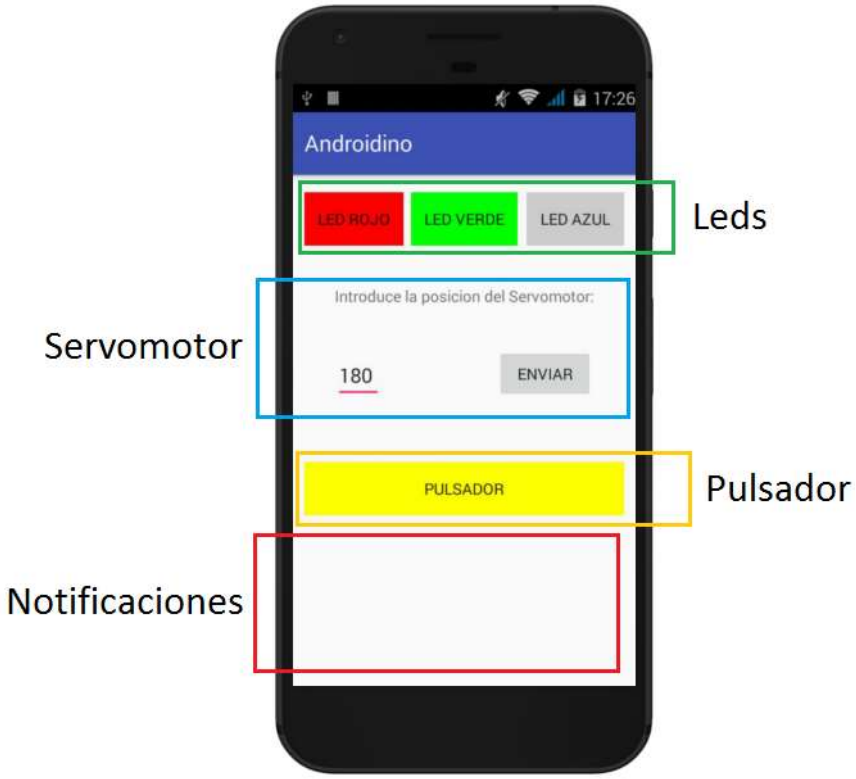


Figura 3-8: Pantalla principal de la app

Leds

En la parte superior de la aplicación se encuentran los botones que permiten el manejo de los leds del sistema. Está formado por tres botones, cada uno correspondiente a uno de los Led.

Cuando un botón es pulsado se comprueba internamente el estado en el que se encuentra dicho led (encendido o apagado) y se abre el proceso para que su estado cambie. El botón reflejara el estado del led mediante su fondo. Estará coloreado si el led se encuentra encendido y será gris si el led está apagado.

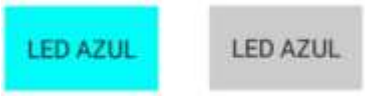


Figura 3-9: Diferencia led activo-inactivo

Servomotor

La aplicación permite controlar la posición de rotación en la que se encuentra el servomotor. Esta posición se basa en el ángulo de giro, por tanto, para indicar la posición se deberá escribir el ángulo (en grados) en el que se quiere situar la pieza. Esta subestructura está formada por tres componentes:

- Cuadro de texto: Donde se insta al usuario a que introduzca la posición numérica del servomotor.

- Campo de introducción: Al pulsarlo se abrirá un teclado numérico para introducir el valor. Una vez introducido el valor permanecerá escrito en la pantalla.

- Botón de enviar: Toma el valor que ha sido inscrito en el campo de introducción y lo envía al servidor para que Arduino lo pueda ejecutar. Antes de enviarlo comprueba que el campo no esté vacío y si lo está mostrara por pantalla una notificación con el problema ocurrido. Tanto el proceso de inserción del valor en la base de datos se ha realizado con éxito como si ha fallado también lo comunicará.

Figura 3-10: Notificaciones del servomotor

Pulsador

El pulsador es un elemento controlado desde el lado de Arduino, por tanto, la aplicación móvil no permite alterar su funcionamiento, solo comprobar cuál es su estado (pulsado o no pulsado). El área del pulsador se coloreará de amarillo si esta pulsado y de gris si no lo está. Además, si se pulsa sobre esta área se verá un mensaje mostrando si el pulsador está activo o no.

Notificaciones

En esta área en blanco es donde se mostrarán por medio de pop-ups notificaciones que sirven para comunicar al usuario cierta información. Hay tres tipos de notificaciones que se muestran:

- Estado: Notifican el estado en el que se encuentra algún componente, por ejemplo, el pulsador.
- Error: Algo no ha salido como se esperaba. Fallo de conexión, error en el servidor…
- Advertencia: Indican al usuario que debe realizar alguna acción. Por ejemplo, rellenar el valor del campo del servomotor antes de enviarla al servidor.

Figura 3-11: Notificaciones de Estado, Error y Advertencia

Capturas

Para finalizar se muestran dos capturas de la aplicación, la de la izquierda es la que se mostrará cuando abramos la app, con todos los elementos desactivados. La de la derecha mostrará todos los elementos activos.

Figura 3-12: Capturas generales de la app

4 Desarrollo

4.1 Servidor

Dado que toda la configuración previa en el servidor ya ha sido realizada por Hostinger, se tratará la parte de los scripts PHP que permiten la comunicación entre un terminal y el propio servidor.

Tanto la app de Android como Arduino se comunicarán con la base de datos mediante la llamada a scripts en PHP. Uno de los principales beneficios que esto tiene es que el código PHP se ejecuta en el lado del servidor, liberando carga de trabajo en el usuario y haciendo que la transferencia de datos entre los terminales y el servidor sea muy reducida.

A la hora de programar se ha buscado la modularidad de las funciones, es decir, dividir el código en diversos archivos.

4.1.1 Scripts PHP Android

A continuación, se analizan los principales scripts de PHP empleados en la transferencia de datos Android-Servidor.

Conexión

Todo archivo PHP que trate de comunicarse con el servidor necesitara abrir una conexión con éste. La conexión se realiza en el archivo *conexión.php* y será requerida en los demás archivos .php. La diferencia entre requerir (función require) e incluir (función include) es que si un script no logra acceder al archivo que se referencia en require no se ejecutara. Si fuese un include, en caso de no encontrarlo, se ejecutaría, produciendo así diversos errores.

```php
<?php

define("servername", "localhost");
define("username", "u469872003_ivan");
define("password", "ComputeR12");
define("dbname", "u469872003_db");

// Create connection

$conn = mysqli_connect(servername, username, password, dbname);

// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}
?>
```

En la variable $conn se guardará la conexión que usarán todos los scripts.

Escritura en la base de datos

Los archivos *on.php, off.php* y *servo.php* realizan la escritura de un dato en la tabla de la base de datos. Todos siguen el mismo patrón que se muestra a continuación.

```
<?php
require 'funciones.php';

$var1 = $_GET['led'];
on($conn,$var1);
?>
```

Se recoge el valor de una variable (en la parte de Android se detallará como se construye la llamada al php y como se pasa la variable) mediante la función GET.
Posteriormente se envía junto a la conexión a una función que realiza la consulta SQL pertinente. Esta función se encuentra en el archivo *funciones.php.*

```
function on($conn2,$color){

$colorled1=$color;
$query1 = "Update RegistrosAnAr Set ";
$query2="= 1 Where identificador = 'Android'";
$queryTOTAL= $query1.$colorled1.$query2;

 // echo($queryTOTAL);
$result1 = mysqli_query($conn2, $queryTOTAL);
}
```

La consulta SQL se construye uniendo diversas cadenas de texto para ser ejecutada finalmente mediante *mysqli_query().*

Para el ejemplo mostrado anteriormente, la consulta SQL tendría un aspecto final así:

Update RegistrosAnAr Set LedR = 1 Where identificador = 'Android'

Siendo *LedR* el led que se quiere encender. Este proceso se lleva a cabo para encender un led, apagarlo y para dar el valor de posición al servo.

Lectura de los valores de la base de datos

La lectura de los valores de la base de datos se lleva a cabo en el script *lectura.php*. Se solicitan los datos de la fila Arduino (ya que se está trabajando en el terminal de Android) con la función *mysqli_query()*. Los datos se guardan en un JSON construido a partir de un array que tiene la siguiente estructura:

{"Tabla"=> $consulta ['Identificador'],
"Rojo"=> $consulta ['LedR'],
"Verde"=> $consulta ['LedV'] ...}

El código completo del script es el siguiente:

```
<?php

require 'conexion.php';

$resultados=mysqli_query($conn,"Select * FROM RegistrosAnAr WHERE Identificador = 'Arduino' ");
while($consulta = mysqli_fetch_array($resultados)){

$miArray = array("Tabla"=> $consulta ['Identificador'],"Rojo"=> $consulta ['LedR'],"Verde"=> $consulta ['LedV'],"Azul"=> $consulta ['LedA'], "Servo"=> $consulta ['Servo'], "Pulsador"=> $consulta ['Pulsador']);

echo(json_encode($miArray));
}
?>
```

4.1.2 Scripts PHP Arduino

Para el script de comunicación de Arduino con el servidor se ha optado por meter todas las acciones en un solo php (*arduin.php*). El principal motivo de esta decisión es que, como se verá a continuación, lanzar la petición para ejecutar el PHP ha de hacerse a través de una petición al servidor por HTTP que es más costoso que ejecutarlo directamente como en Android.

La primera parte del script recoge el valor de las variables como lo hacían *on.php, off.php* y *servo.php* y las guarda en la fila de Arduino.

```
$var1 = $_GET['ledR'];
$var2 = $_GET['ledV'];
$var3 = $_GET['ledA'];
$var4 = $_GET['pulsador'];

$query1 = "Update RegistrosAnAr Set LedR = ";
$query2 = ", LedV = ";
$query3 = ", LedA = ";
$query4 = ", Pulsador = ";
$query5 = " Where identificador = 'Arduino'";

$QueryFinal = $query1.$var1.$query2.$var2.$query3.$var3.$query4.$var4.$query5;
$result1 = mysqli_query($conn, $QueryFinal);
```

En la segunda parte se hace la consulta SQL para recibir los datos de Android. Al igual que antes, se construye un JSON que será tratado por Arduino.

```
$resultados=mysqli_query($conn,"Select * FROM RegistrosAnAr WHERE Identificador
= 'Android' ");
while($consulta = mysqli_fetch_array($resultados)){

$miArray = array("Tabla"=> $consulta ['Identificador'],"Rojo"=> $consulta
['LedR'],"Verde"=> $consulta ['LedV'],"Azul"=> $consulta ['LedA'], "Servo"=> $consulta
['Servo'], "Pulsador"=> $consulta ['Pulsador']);

echo(json_encode($miArray));
}
```

4.2 Arduino

Para realizar la programación de Arduino se ha utilizado el software oficial que se puede descargar gratuitamente en la página oficial de la placa.

La estructura del programa puede dividirse en tres funciones:

- **Void setup().** Se ejecuta al comenzar el programa una única vez.
- **Void leer().** Encargada de realizar la petición php.
- **Void loop().** Se ejecuta continuamente mientras Arduino se encuentre encendido.

Antes de definir estas tres funciones se declaran las librerías que se vayan a utilizar y las variables necesarias.

En cuanto a las librerías, serán tres. *SPI.h* empleada para poder utilizar el puerto Serial de Arduino. *Servo.h* para controlar el servomotor. *Ethernet2.h* para utilizar todas las funcionalidades de la placa Ethernet. Esta última librería no viene instalada por defecto ya que la que viene es la librería *Ethernet.h* empleada en la anterior revisión del periférico. Una vez la instalemos con el asistente de librerías incluido en el programa, su uso será exactamente el mismo que el de la librería *Ethernet.h*.

Respecto a las variables que vamos a utilizar, vamos a ver las más destacables:

```
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

char server[] = "www.actuadorandroid.hol.es";

IPAddress ip(192, 168, 1, 50);

EthernetClient client;

Servo servoMotor;
```

La primera variable se corresponde a la dirección MAC. El valor es el recomendado por Arduino.

La segunda variable es la dirección del servidor con el que vamos a contactar. La conexión también podría realizarse mediante la dirección IP.

La tercera variable es la dirección IP que tendrá Arduino dentro de la subred a la que lo conectamos. Para saber que IP podemos asignarle, usando una terminal vemos la dirección IP que tenga un ordenador conectado a la subred. Los tres primeros números se mantienen y para el cuarto emplearemos un número libre dentro de la subred. Para saber si la IP escogida es válida basta con hacer un Ping a la dirección.

Las dos últimas variables inicializan un cliente Ethernet y el servomotor.

4.2.1 Setup

La función comienza con la inicialización de los pines empleados y del bus Serial.
El pin 8 es la entrada del pulsador, Los pines 3, 4 y 5 de salida para activar los Leds y el 9 el que controlara al servo.

```
  pinMode(8, INPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);

 servoMotor.attach(9);

  Serial.begin(9600);
```

Para finalizar iniciamos la conexión Ethernet. Si falla en primera instancia se volverá a intentar a través de la IP.

```
if (Ethernet.begin(mac) == 0) {
    Serial.println("Fallo al configurar Ethernet");
    // Inicialización alternativa por IP
    Ethernet.begin(mac, ip);
```

A lo largo del código será recurrente ver la instrucción Serial.print/Serial.println que imprime por pantalla a través del puerto serial un mensaje. Su uso no es necesario, pero sí que ayuda a la hora de programar y debuggear el código.

4.2.2 Leer

La función *leer()* Se encarga de enviar una petición al servidor. La petición es de tipo GET y se realiza siguiendo el protocolo HTTP. El GET se divide en secciones para poder rellenar las variables con los valores obtenidos por Arduino. La respuesta a esta petición se analizará en la función *loop()*.

```
if (client.connect(server, 80)==1) {
    Serial.println("Se ha conectado al servidor");

    client.print("GET /arduin.php?ledR=");
    client.print(LedR);
    client.print("&ledV=");
    client.print(LedV);
    client.print("&ledA=");
    client.print(LedA);
    client.print("&pulsador=");
    client.print(estadoPulsador);
    client.print(" HTTP/1.1\r\n");
    client.print("Host: www.actuadorandroid.hol.es \r\n\r\n");
    //client.println();
    Serial.println("Peticion enviada");
```

4.2.3 Loop

La función *loop()* comienza leyendo en un bucle la respuesta HTTP que nos da el servidor. Se lee carácter a carácter y se almacena la respuesta en el String *codigo*.

```
while (client.available()) {
  char c = client.read();
  Serial.print(c);
  codigo += c;
}
```

La respuesta que envía el servidor siempre tiene el mismo formato, que es el siguiente:

```
Conectando...
Se ha conectado al servidor
Peticion enviada
HTTP/1.1 200 OK
Date: Wed, 21 Jun 2017 18:43:30 GMT
Server: Apache
X-Powered-By: PHP/5.6.21
Content-Length: 81
Content-Type: text/html; charset=UTF-8

{"Tabla":"Android","Rojo":"1","Verde":"0","Azul":"1","Servo":"25","Pulsador":"0"}HTTP/1.1 200 OK
```

Figura 4-1: Respuesta HTTP

Como el final de la respuesta siempre es el JSON se obtendrán los datos seleccionando la posición que ocupa cada respuesta que queremos empezando por el final.

```
int num = codigo.length();
rojo= num -54;
verde= num -42;
azul= num -31;

LedR=codigo[rojo];
LedV=codigo[verde];
LedA=codigo[azul];
Ser1=codigo[servo1];
Ser2=codigo[servo2];
```

Para el servo se necesitan dos dígitos y transformar los caracteres en tipo int para escribir su posición. Restando 48 a un char obtenemos el número int (Tabla ASCII). Para obtener el número final multiplicamos las decenas por diez y sumamos las unidades.
Después escribimos la posición del servo con la función write().

```
int a = Ser1-48;
int b = Ser2-48;
int c = a*10+b;

microServo.write(c);
```

Posteriormente se asigna el valor del pulsador, según este activo/inactivo y se activan las salidas para encender los leds si así lo indica el servidor.

```
if (pulsador==HIGH){
    estadoPulsador = 1; }

else{ estadoPulsador = 0;}

if (LedR=='1'){
   digitalWrite(3, HIGH);}

else{ digitalWrite(3, LOW);}
```

Para finalizar, cuando se detecta que el servidor ha terminado la comunicación HTTP se cierra el cliente y tras pausar el programa se vuelve a lanzar la función leer que vuelve a enviar la petición GET al servidor.

```
  if (!client.connected()) {
   Serial.println();
    Serial.println("disconnecting.");
    client.stop();
    delay(200);

    leer();
    delay (2000);
  }
```

4.3 Android

Veremos ahora como ha sido el desarrollo de la App para Android, entrando en los detalles del código que la componen.

4.3.1 Conceptos generales

Pese a que Android Studio ofrece varias plantillas predefinidas, se ha optado por empezar el proyecto desde cero, con una plantilla vacía.

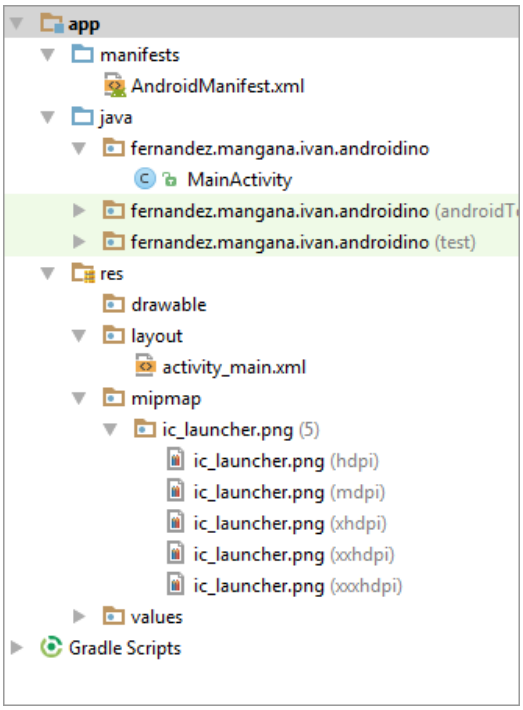


Figura 4-2: Estructura del proyecto Android Estudio

A la izquierda se puede ver una imagen con la estructura del proyecto. Las partes más importantes son las siguientes:

- **AndroidManifest.xml:** Indican aspectos externos de la aplicación, por ejemplo los permisos especiales que necesita.
- **MainActivity:** Al ser una aplicación con una sola pantalla todo el código java de la aplicación irá en este archivo.
- **Activity_main.xml:** La parte gráfica de la aplicación se encuentra definida en este archivo. Elementos como los botones, cuadros de texto…
- **Ic_launcher.png:** Se encuentran alojadas las imágenes del icono de nuestra aplicación.

La manera en la que un usuario interactúa con la aplicación es a través de los botones de la aplicación. Cuando un botón es pulsado es detectado por la función *onClick*. Esta función obtiene el identificador del botón que ha sido pulsado y ejecuta las acciones específicas dentro de un switch. A continuación, se puede ver un fragmento de esta función:

```
public void onClick(View v) {
    switch (v.getId()){

        case R.id.LRojo:
            if (LRojoL.equals("0")){
                hiloActivar = new Activar();
                hiloActivar.execute(Encender,"LedR");
                break;
            }
            if(LRojoL.equals("1")){
                hiloActivar = new Activar();
                hiloActivar.execute(Apagar,"LedR");
                break;
            }
            break;
```

Nota: Aspectos como la definición de los componentes, la definición de variables, activación de botones etc. no se detallarán en la memoria por su carácter universal independiente al proyecto.

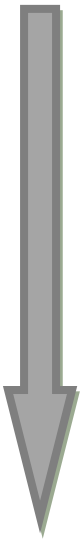
4.3.2 AsyncTask

Una de las principales diferencias entre realizar la programación para una versión de API 15 o superior (este caso) y una versión anterior radica en la manera de realizar la conexión a internet. Mientras que en las versiones anteriores se podía establecer una conexión a internet desde el hilo principal, en estas nuevas versiones es necesario lanzar un segundo hilo para estas conexiones. Este hilo se ejecuta en paralelo al principal comunicándose con él cuando sea necesario.

Este nuevo hilo queremos que se ejecute de manera asíncrona, ya que cuando se producen los intercambios de datos a través de internet no es deseable que la aplicación quede congelada e interrumpa su funcionamiento hasta terminar. Es por ello que se emplea la extensión de AsyncTask para su implementación.

El núcleo principal de AsyncTask se lleva a cabo en la función doInBackground. En esta parte no se puede interactuar con el hilo principal. Este hilo principal es el encargado de ejecutar y administrar la parte grafica de la aplicación. Por tanto, en la función principal de AsyncTask no se podrán realizar cambios en la pantalla del dispositivo. Para ello se ejecutan otra serie de funciones dentro del propio AsyncTask.

Las funciones principales de AsyncTask, ordenadas por orden de ejecución, son las siguientes:



- **onPreExecute:** Se ejecuta antes de que el hilo sea lanzado. Estas tareas se llevaran a cabo en el hilo principal permitiendo alterar la interfaz de usuario.
- **onProgressUpdate:** Se ejecuta mientras el hilo secundario está en marcha, permitiendo interactuar con el hilo principal.
- **doInBackground:** Es el núcleo del hilo AsyncTask. En él se ejecutan todas las acciones principales del hilo.
- **OnPostExecute:** Se ejecuta después de que la función anterior haya terminado interactuando con el hilo principal.
- **onCancelled:** En esta función se definen los métodos de actuación que se quieran realizar si el hilo es cancelado por algún motivo.

En la app se emplearán dos hilos secundarios que extienden a AsyncTask, uno para la lectura de datos del servidor y otro para la escritura. Sus principales características se verán a continuación.

4.3.3 Hilo de escritura

El hilo de escritura es ejecutado cuando el usuario pulsa un botón que realiza la acción de escribir un dato en la base de datos. Al pulsarlo se lanzará el hilo (switch de la función *onClick*) con la siguiente sentencia:

hiloActivar= new Activar();
hiloActiar.execute(Parametro[0], Parametro[1]);

Al hilo se le pasan dos parámetros. El primero hace referencia al tipo de acción que se llevara a cabo. Hay tres tipos:

- **Encender:** Activa un led del sistema.
- **Apagar:** Apaga un led del sistema.
- **Servo:** Indica la posición del servomotor.

El segundo parámetro se emplea solo en las funciones Encender y Apagar. Sirve para indicar cuál de los tres leds del sistema es sobre el que se quiere ejecutar la acción.

Dentro de la función *doInBackground* dentro del hilo se llevará a cabo la parte principal. Se muestran como ejemplo las acciones para indicar al servidor que queremos encender un determinado led (Apagar y Servo siguen exactamente el mismo patrón):

```
if (params[0] == Encender) {
    try {
        Envio= "?led=" + params[1];

        URL url = new URL(IP + Encender + Envio);
        HttpURLConnection connection = (HttpURLConnection)
url.openConnection(); //abrir conexion
        connection.setRequestProperty("User-Agent", "Mozilla/5.0" +
"(Linux; Android 4.0; es-ES) Ejemplo HTTP");

        int respuesta = connection.getResponseCode();
        if (respuesta == HttpURLConnection.HTTP_OK) {
            error=2;
        }

    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Se utiliza la clase *Url* propia de java. El primer paso es construir la cadena que formara la propia url. Para eso interconectamos distintas subcadenas.

IP + Encender + Envio

- **IP** se corresponde con la dirección de la web alojada en el servidor (http://actuadorandroid.hol.es/).
- **Encender** tiene el archivo PHP que se tiene que ejecutar para llevar a cabo la acción. En este caso es *on.php*.

- **Envío** para el caso de Encender y Apagar contiene el led sobre el que se actua. Para Servo contiene la posición. La manera de pasar estas variables a los archivos PHP es a través de la propia url.

Después de construir la url se le añade una cabecera Http. No es estrictamente necesario, pero solventa posibles problemas que puedan llegar a ocurrir. Si la comunicación ha sido realizada con éxito se cambia el valor de la variable error. Según el valor de esta variable, en la función *OnPostExecute* que se ejecuta al terminar el hilo, mostrara mensajes de error o no.

Todo el proceso que acabamos de mencionar ha de ser ejecutado dentro de un Try-Catch ya que la clase *Url* genera varias excepciones.

4.3.4 Hilo de lectura

El hilo de lectura arranca cuando se inicia la aplicación y se ejecuta continuamente ya que es necesario tener los valores de los componentes en todo momento. La primera parte de este hilo es exactamente igual al hilo de escritura en el que se construye la url y se envía la petición HTTP. Pasemos a ver las diferencias que vienen a continuación.

Una vez que se envía la petición, si todo ha ido correctamente se recibirá una respuesta. La respuesta es el objeto JSON que se explicó en la parte de código PHP en el servidor (4.1.1).

Recordemos la estructura del JSON:

```
{"Tabla"=> $consulta ['Identificador'],
"Rojo"=> $consulta ['LedR'],
"Verde"=> $consulta ['LedV'],
"Azul"=> $consulta ['LedA'],
"Servo"=> $consulta ['Servo'],
"Pulsador"=> $consulta ['Pulsador']};
```

Esta respuesta se almacenará en un buffer de entrada y será parseado para descomponerlo en distintas partes. Se separará la cadena utilizando /"/ y se asignan las posiciones adecuadas a cada variable correspondiente a los elementos. Se ha realizado de esta manera porque al haber pocos elementos es más sencillo. Si la cantidad de elementos fuese mucho mayor, podría hacerse la asignación a través de los identificadores de los campos: Rojo, Servo…

```
if (respuesta == HttpURLConnection.HTTP_OK) {

    BufferedReader lectura = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
    String linea = lectura.readLine();

    String[] valoresLed = linea.split("\"");
    LRojoL=valoresLed[7];
    LVerdeL=valoresLed[11];
    LAzulL=valoresLed[15];
    PulsadorL=valoresLed[23];
}
```

Una vez realizada la lectura se pasa a la función *OnPostExecute* donde se colorearán los leds y el pulsador de acuerdo al valor recogido de la base de datos con esta sentencia:

```
if(LRojoL.equals("1")){LRojo.setBackgroundColor(Color.RED);}
else if (LRojoL.equals("0")){LRojo.setBackgroundColor(Color.LTGRAY);}
```

Para finalizar se duerme el hilo durante un segundo y se vuelve a lanzar

```
try {
    Thread.sleep(1000);
hiloAutomatico = new Automatico();
hiloAutomatico.execute(Valor,"0");

} catch(InterruptedException e) {}
```

El motivo de dormir el hilo es para que no se ejecute continuamente y liberar así algo de carga al servidor. Al estar un segundo dormido se producirá un retraso adicional máximo de un segundo en la muestra del cambio de estado de un componente. Es decir, si se apaga un led justo en el momento en el que el hilo está dormido, el usuario verá el cambio en la interfaz con un segundo de retraso. Si el tiempo de respuesta de la aplicación fuera un parámetro crítico se podría quitar este punto y lanzar de nuevo el hilo sin pararlo.

4.3.5 Permisos

El único permiso especial que ha de tener la app es el de uso de internet. Este permiso es común tanto para redes Wifi como para redes de telefonía. Es otorgado en el AndroidManifest.xml mediante el siguiente código:

```
<uses-permission android:name="android.permission.INTERNET">
```

4.3.6 Toast

Para mostrar al usuario los distintos mensajes informativos (vistos en la parte de Diseño) se harán uso de la clase Toast que permite lanzar pop-ups en la pantalla de la aplicación. Aquí podemos ver su estructura básica. Todos los Toast del sistema tienen una duración corta.

```
Toast.makeText(getApplicationContext(),"Se ha producido un error",
Toast.LENGTH_SHORT).show();
```

5 Integración, pruebas y resultados

Se analizarán ahora las pruebas que se han llevado a cabo para asegurarnos el correcto funcionamiento del sistema, el tratamiento de errores y la integración final.

5.1 Control de errores

Del lado de la App de Android se ha tenido en cuenta varios aspectos para evitar todos los posibles fallos que se pueden presentar:

- Cada componente al que se quiere modificar su estado lanza su propio hilo individualmente. Se permite así que aunque se pulse un componente detrás de otro sin ninguna pausa intermedia se actualice correctamente el estado.

- Se ha limitado la entrada numérica para que no pueda ser introducido ningún carácter que no sea un número en la posición del servomotor.

- Se muestra un mensaje indicando el tipo de error ocurrido para que el usuario sepa que es lo que ha fallado.

- Las pruebas se han realizado tanto por conexión WIFI como por medio de la telefonía móvil 3G.

Del lado de Arduino se han tomado las siguientes medidas:

- Cada componente (leds, pulsador, servomotor) se ha probado individualmente para asegurarse de que su funcionamiento es correcto.

- La placa de Arduino se ha cargado con programas preestablecidos facilitados por el fabricante para verificar su correcto funcionamiento. El mismo proceso ha sido utilizado con la tarjeta de expansión de Ethernet.

- Se han utilizado las pausas (delay) en el programa recomendadas por el fabricante para no saturar la placa

Cada parte del proyecto ha sido verificada individualmente para comprobar que su funcionamiento era correcto. Para ello se ha abierto la base de datos y manualmente se han ido alterando los campos para verificar como se comportaba cada uno de los dos terminales por separado. Todas las pruebas realizadas han sido positivas.

5.2 Realización de pruebas

Una vez desarrollado todo el sistema se ha optado por elaborar un Test-Case con diferentes pruebas. Estas pruebas se recogen en la siguiente tabla. Cada una ha sido repetida en un total de 10 ocasiones. Se ha recogido también el tiempo medio que ha transcurrido entre realizar una acción y ver el resultado tanto en Android como en Arduino.

	Superado	Tiempo Android	Tiempo Arduino
Encender Led Rojo	Si	7.200s	5.720s
Encender Led Verde	Si	7.450s	5.430s
Encender Led Azul	Si	7.260s	5.620s
Apagar Led Rojo	Si	6.900s	5.480s
Apagar Led Verde	Si	6.780s	4.800s
Apagar Led Azul	Si	7.230s	5.270s
Activar pulsador	Si	4.290s	No procede
Posicionar Servo	Si	No procede	3.800s
Cambiar estado 3 leds simultaneos (on, off on)	Si	7.310s	5.810s
Cambiar estado 3 leds simultaneos (on, on off)	Si	7.440s	5.640s
Activar pulsador + led Verde	Si	7.310s	5.540s
Activar pulsador + posicionar Servo	Si	4.870s	4.370s

Tabla 5-3: Resultados de la evaluación

Antes de entrar a valorar los resultados obtenidos hay que comentar un aspecto relevante del servidor. Al ser gratuito solo permite 60 peticiones por minuto de parte de un dispositivo, hasta un máximo de 250 peticiones por minuto totales. Por este hecho se duerme el hilo de lectura de Android durante un segundo y el de Arduino dos segundos (uno por lectura más uno para no saturar la placa).

Realizando una prueba rápida eliminando este tiempo de pausa en Android y bajando el tiempo de Arduino hasta el medio segundo, se obtienen unos resultados significativamente menores. Estas pruebas no han sido tan extensas como las anteriores por los motivos ya comentados.

	Superado	**Tiempo Android**	**Tiempo Arduino**
Encender Led Rojo	Si	5.360s	3.630s
Encender Led Verde	Si	5.170s	3.970s
Apagar Led Azul	Si	5.440s	3.730s
Activar pulsador	Si	3.310s	No procede

Tabla 5-4: Resultados de la evaluación disminuyendo retardos

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Tras analizar los resultados del apartado anterior, se puede concluir que los objetivos preestablecidos al comienzo de esta memoria han sido cumplidos con éxito. Se ha conseguido tener un sistema estable de comunicación entre los dos terminales a partir del cual se pueden llevar a cabo múltiples aplicaciones y que puede ser ampliado en numerosas direcciones.

Tanto la parte de Android como la de Arduino se comunican correctamente con la base de datos del servidor. Este último aspecto, el del servidor, actúa como cuello de botella en la comunicación (como se vio en el apartado 5) y simplemente contratando uno de pago se puede llegar a conseguir unos retardos más bajos.

A lo largo del texto se ha ido comentando todo el detalle del proceso, permitiendo que una persona totalmente ajena al proyecto sea capaz de entenderlo y poder continuar su desarrollo y aplicación.

Se concluye entonces con la satisfacción de haber logrado los objetivos perseguidos y con la profundización de los conocimientos en múltiples campos vistos durante el grado como los servicios web, los sistemas embebidos o lenguajes de programación como php y java.

6.2 Trabajo futuro

Respecto al trabajo futuro, se pueden apreciar dos corrientes claras.

Por un lado tenemos las aplicaciones directas del sistema de comunicación. Con aplicaciones nos referimos a la implantación del sistema de comunicación en proyectos de uso real. El primer ejemplo que se puede venir a la cabeza es el de la domótica. Control de luces, persianas, pequeños electrodomésticos…

Pero existen otros muchos usos como por ejemplo el de realizar juegos de tablero (tres en raya, damas, ajedrez…) entre un terminal móvil y un tablero físico controlado con el procesador embebido. O, por qué no, crear aplicaciones sencillas para que un niño aprenda un lenguaje de programación de manera visual.

El otro enfoque para para continuar expandiendo el proyecto es el de reemplazar un terminal por otro. Recordemos que dado el diseño realizado, sustituyendo uno de los terminales el otro puede continuar funcionando sin tener que modificarlo.

Se podría cambiar el terminal móvil por otro con un sistema operativo diferente. O Cambiar la placa Arduino por otras que se han visto en el apartado de estado del arte, como Raspberry o Nanode.

Referencias

[1] Roberto Montero Miguel, “Desarrollo de aplicaciones para Android”, 2012.

[2] Joan Ribas Lequerica, “Desarrollo de aplicaciones para Android”, 2016.

[3] Daniel Lozano Equisoain, “Arduino Práctico”, 2016.

[4] José Rafael Lajara Vizcaíno, José Pelegrí Sebastiá, “Sistemas integrados con Arduino”, 2013.

[5] Timothy Boronczyk y Martin E. Psinas, “PHP y MySQL”, 2009.

[6] Edgar DÁndrea, “PHP5”, 2005.

[7] Kostadin Nedeltchev Koroutchev, “Transparencias de la asignatura: Sistemas Distribuidos, Tema 2: SQL”, 2017.

[8] Andri Yadi, Ria Sri Rahayu, “Project Gallon: https://www.hackster.io/”, 2016.

[9] Johanmoberg, “App and Timer Controlled Automatic Blinds: http://www.instructables.com”, 2015.

[10] Página web de Arduino: https://www.arduino.cc/, 2017.